Basics

Basic syntax from the python programming language

Showing Output To User

the print function is used to display or print output

```python
print("Content that you wanna print on screen")
```

Taking Input From User

the input function is used to take input from the user

```python
var1 = input("Enter your name: ")
```

Empty List

This method allows you to create an empty list

```python
my_list = []
```

Empty Dictionary

By putting two curly braces, you can create a blank dictionary

```python
my_dict = {}
```

Range Function

range function returns a sequence of numbers, eg, numbers starting from 0 to n-1 for range(0, n)

```python
range(int_value)
```

Comments

Comments are used to make the code more understandable for programmers, and they are not executed by compiler or interpreter.

Single line comment

```
#This is a single line comment
```

Multi-line comment

```
'''This is a
multi-line
comment'''
```

Escape Sequence

An escape sequence is a sequence of characters; it doesn't represent itself when used inside string literal or character.

Newline

Newline Character

```
\n
```

Backslash

It adds a backslash

```
\\
```

Single Quote

It adds a single quotation mark

```
\'
```

Tab

It gives a tab space

```
\t
```

Backspace

It adds a backspace

```
\b
```

Octal value

It represents the value of an octal number

```
\ooo
```

Hex value

It represents the value of a hex number

```
\xhh
```

Carriage Return

Carriage return or \r is a unique feature of Python. \r will just work as you have shifted your cursor to the beginning of the string or line.

```
\r
```

Strings

Python string is a sequence of characters, and each character can be individually accessed. Using its index.

String

You can create Strings by enclosing text in both forms of quotes - single quotes or double-quotes.

```
variable_name = "String Data"
```

Slicing

Slicing refers to obtaining a sub-string from the given string.

```
var_name[n : m]
```

String Methods isalnum() method

Returns True if all characters in the string are alphanumeric

```
string_variable.isalnum()
```

isalpha() method

Returns True if all characters in the string are alphabet

```
string_variable.isalpha()
```

isdecimal() method

Returns True if all characters in the string are decimals

```
string_variable.isdecimal()
```

isdigit() method

Returns True if all characters in the string are digits

```
string_variable.isdigit()
```

islower() method

Returns True if all characters in the string are lower case

```
string_variable.islower()
```

isspace() method

Returns True if all characters in the string are whitespaces

```
string_variable.isspace()
```

isupper() method

Returns True if all characters in the string are upper case

```
string_variable.isupper()
```

lower() method

Converts a string into lower case

```
string_variable.lower()
```

upper() method

Converts a string into upper case

```
string_variable.upper()
```

strip() method

It removes leading and trailing spaces in the string

```
string_variable.strip()
```

List

A List in Python represents a list of comma-separated values of any data type between square brackets.

List

```
var_name = [element1, element2, and so on]
```

List Methods index method

Returns the index of the first element with the specified value

```
list.index(element)
```

append method

Adds an element at the end of the list

```
list.append(element)
```

extend method

Add the elements of a list (or any iterable) to the end of the current list

```
list.extend(iterable)
```

insert method

Adds an element at the specified position

```
list.insert(position, element)
```

pop method

Removes the element at the specified position and returns it

```
list.pop(position)
```

remove method

The remove( ) method removes the first occurrence of a given item from the list

```
list.remove(element)
```

clear method

Removes all the elements from the list

```
list.clear()
```

count method

Returns the number of elements with the specified value

```
list.count(value)
```

reverse method

Reverse the order of the list

```
list.reverse()
```

sort method

Sorts the list

```
list.sort(reverse=True|False)
```

Tuples

Tuples are represented as a list of comma-separated values of any data type within parentheses.

Tuple Creation

```
variable_name = (element1, element2, ...)
```

Tuple Methods count method

It returns the number of times a specified value occurs in a tuple

```
tuple.count(value)
```

index method

It searches the tuple for a specified value and returns the position.

```
tuple.index(value)
```

Sets

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

Set Creation: Way 1

```
var_name = {element1, element2, ...}
```

Set Creation: Way 2

```
var_name = set([element1, element2, ...])
```

Set Methods: add() method

Adds an element to a set

```
set.add(element)
```

clear() method

Remove all elements from a set

```
set.clear()
```

discard() method

Removes the specified item from the set

```
set.discard(value)
```

intersection() method

Returns intersection of two or more sets

```
set.intersection(set1, set2 ... etc)
```

issubset() method

Checks if a Set is Subset of Another Set

```
set.issubset(set)
```

pop() method

Removes an element from the set

```
set.pop()
```

remove() method

Removes the specified element from the Set

```
set.remove(item)
```

union() method

Returns the union of Sets

```
set.union(set1, set2...)
```

Dictionaries

The dictionary is an unordered set of comma-separated key: value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

Dictionary

```
<dictionary-name> = {<key>: value, <key>: value ...}
```

Adding Element to a dictionary

By this method, one can add new elements to the dictionary

```
<dictionary>[<key>] = <value>
```

Updating Element in a dictionary

If the specified key already exists, then its value will get updated

```
<dictionary>[<key>] = <value>
```

Deleting Element from a dictionary

del let to delete specified key: value pair from the dictionary

```
del <dictionary>[<key>]
```

Dictionary Functions & Methods len() method

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary

```
len(dictionary)
```

clear() method

Removes all the elements from the dictionary

```
dictionary.clear()
```

get() method

Returns the value of the specified key

```
dictionary.get(keyname)
```

items() method

Returns a list containing a tuple for each key-value pair

```
dictionary.items()
```

keys() method

Returns a list containing the dictionary's keys

```
dictionary.keys()
```

values() method

Returns a list of all the values in the dictionary

```
dictionary.values()
```

update() method

Updates the dictionary with the specified key-value pairs

```
dictionary.update(iterable)
```

Conditional Statements

The if statements are the conditional statements in Python, and these implement selection constructs (decision constructs).

if Statement

```
if(conditional expression):
statements
```

if-else Statement

```
if(conditional expression):
statements
else:
statements
```

if-elif Statement

```
if (conditional expression) :
statements
elif (conditional expression) :
statements
else :
statements
```

Nested if-else Statement

```python
if (conditional expression):
if (conditional expression):
statements
else:
statements
else:
statements
```

Iterative Statements

An iteration statement, or loop, repeatedly executes a statement, known as the loop body, until the controlling expression is false (0).

For Loop

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.

```python
for <variable> in <sequence>:
statements_to_repeat
```

While Loop

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remains true.

```python
while <logical-expression> :
loop-body
```

Break Statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

```python
for <var> in <sequence> :
statement1
if <condition> :
break
statement2
statement_after_loop
```

Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

```
for <var> in <sequence> :
statement1
if <condition> :
continue
statement2
statement3
statement4
```

Functions

A function is a block of code that performs a specific task. You can pass parameters into a function. It helps us to make our code more organized and manageable.

Function Definition

```
def my_function(parameters):
# Statements
```

File Handling

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open() function

```
var_name = open("file name", "opening mode")
```

close() function

```
var_name.close()
```

Read () function

The read functions contains different methods, read(),readline() and readlines()

```
read() #return one big string
```

It returns a list of lines

```
read-lines
```

It returns one line at a time

```
readline
```

Write () function

This function writes a sequence of strings to the file.

```
write () #Used to write a fixed sequence of characters to a file
```

It is used to write a list of strings

```
writelines()
```

Append () function

The append function is used to append to the file instead of overwriting it. To append to an existing file, simply open the file in append mode (a):

```
file = open("Hello.txt", "a")
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try and except

A basic try-catch block in python. When the try block throws an error, the control goes to the except block.

```
try:
[Statement body block]
raise Exception()
```

```
except Exception as e:
[Error processing block]
```

OOPS

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in python

```
class class_name:
#Statements
```

class with a constructor

The syntax for writing a class with the constructor in python

```
class CodeWithHarry:

# Default constructor
def __init__(self):
self.name = "CodeWithHarry"

# A method for printing data members
def print_me(self):
print(self.name)
```

object

Instantiating an object

```
<object-name> = <class-name>(<arguments>)
```

filter function

The filter function allows you to process an iterable and extract those items that satisfy a given condition

```
filter(function, iterable)
```

issubclass function

Used to find whether a class is a subclass of a given class (classinfo) or not

```python
issubclass(class, classinfo)
```

Iterators and Generators

Here are some of the advanced topics of the Python programming language like iterators and generators

Iterator

Used to create an iterator over an iterable

```python
iter_list = iter(['Harry', 'Aakash', 'Rohan'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Generator

Used to generate values on the fly

```python
# A simple generator function
def my_gen():
n = 1
print('This is printed first')
# Generator function contains yield statements
yield n
n += 1
print('This is printed second')
yield n
n += 1
print('This is printed at last')
yield n
```

Decorators

Decorators are used to modifying the behavior of function or class. They are usually called before the definition of a function you want to decorate.

property Decorator (getter)

```
@property
def name(self):
return self.__name
```

setter Decorator

It is used to set the property 'name'

```
@name.setter
def name(self, value):
self.__name=value
```

Deletor Decorator

It is used to delete the property 'name'

```
@name.deleter #property-name.deleter decorator
def name(self, value):
print('Deleting..')
del self.__name
```